Name: ______________________________

Rules and Hints

- You may use one handwritten 8.5 x 11" cheat sheet (front and back). This is the only additional resource you may consult during this exam. *No calculators.*
- When you write code, be sure that the indentation level of each statement is clear.
- Explain/show work if you want to receive partial credit for wrong answers.
- As long as your code is correct, you will get full credit. No points for style.
- As always, the SSU rules on academic integrity are in effect.

Problem	**Max Score**	**Your Score**
Problem 1: Binary Search	10	
Problem 2: Selection Sort	10	
Problem 3: Mergesort	10	
Problem 4: Trace Functions	20	
Problem 5: Defining classes	20	
Problem 6: Using classes	30	
Total	100	

Cheat Sheet Additions

The functions below are for your reference on Problems 1 and 2.

```
def binary_search(search_list, value_to_find):
    first = 0
    last = len(search_list) - 1

    while first <= last:
        middle = (first + last) // 2
        # Problem 1: state the values of first, last,
        # and middle at this point in the code
        if value_to_find == search_list[middle]:
            return middle
        elif value_to_find < search_list[middle]:
            last = middle - 1
        else:
            first = middle + 1
    return None

def selection_sort(list_to_sort):
    for i in range(len(list_to_sort) - 1):
        min_index = find_min_index(list_to_sort, i)
        swap(list_to_sort, i, min_index)
        # Problem 2: Show list contents at this point

def swap(L, i, j):
    x = L[i]
    L[i] = L[j]
    L[j] = x

def find_min_index(L, s):
    min_index = s
    for i in range(s, len(L)):
        if L[i] < L[min_index]:
            min_index = i
    return min_index
```

Cheat Sheet Additions

The functions below are for your reference on Problem 3.

```
def merge(L, start_index, sublist_size):
    index_left = start_index
    left_stop_index = start_index + sublist_size
    index_right = start_index + sublist_size
    right_stop_index = min(start_index + 2 * sublist_size, len(L))
    L_tmp = []

    while (index_left < left_stop_index and
           index_right < right_stop_index):

        if L[index_left] < L[index_right]:
            L_tmp.append(L[index_left])
            index_left += 1
        else:
            L_tmp.append(L[index_right])
            index_right += 1

    if index_left < left_stop_index:
        L_tmp.extend(L[index_left : left_stop_index])
    if index_right < right_stop_index:
        L_tmp.extend(L[index_right : right_stop_index])

    L[start_index : right_stop_index] = L_tmp

def merge_sort(L):
    chunksize = 1
    while chunksize < len(L):
        left_start_index = 0 # Start of left chunk in each pair

        while left_start_index + chunksize < len(L):
            merge(L, left_start_index, chunksize)
            left_start_index += 2 * chunksize

        chunksize *= 2
        # Problem 3: Show list contents at this point
```

Problem 1: **Binary Search (10 points)**

Consider the following sorted list:

```
L = ['drax',
     'groot',
     'natasha',
     'nebula',
     'rhodey',
     'steve',
     'tchalla',
     'thor',
     'vision',
     'wanda']
```

Problem 1A Fill out the below table, tracing the call `v = binary_search(L, 'nebula')`, a binary search for `'nebula'` in this list. Fill out one row per iteration of the loop (per the comment in the code on page 2). If there are more rows than iterations, leave the extra rows blank. At the end, write the value `v` returned by the function.

Iteration	Value of `first`	Value of `last`	Value of `middle`	Value of `L[middle]`
1				
2				
3				
4				
5				

Return value `v`: __________

Problem 1B Fill out the below table, tracing the call `v = binary_search(L, 'wanda')`, a binary search for `'wanda'` in this list. Fill out one row per iteration of the loop (per the comment in the code on page 2). If there are more rows than iterations, leave the extra rows blank. At the end, write the value `v` returned by the function.

Iteration	Value of `first`	Value of `last`	Value of `middle`	Value of `L[middle]`
1				
2				
3				
4				
5				

Return value `v`: __________

Problem 2: **Selection Sort (10 points)**

Consider the following list:

```
L = ['turley',
     'starkey',
     'noodler',
     'jukes',
     'smee',
     'mullins',
     'whibbles',
     'cecco']
```

In the table below, show the *contents* of the list after each of the first four iterations of the for-loop in `selection_sort` (per the comment in the code on page 2).

You may just draw a horizontal line between cells if a word has *not* changed position.

Index	Initial Order	After $i = 0$ iteration	After $i = 1$ iteration	After $i = 2$ iteration	After $i = 3$ iteration
0	turley				
1	starkey				
2	noodler				
3	jukes				
4	smee				
5	mullins				
6	whibbles				
7	cecco				

Problem 3: **Mergesort (10 points)**

Consider the following list:

```
L = ['smee',
     'noodler',
     'turley',
     'cecco',
     'jukes',
     'mullins',
     'whibbles',
     'starkey']
```

In the diagrams below, show the contents of the list after each of the first three iterations of the outer while-loop in `merge_sort` (per the comment in the code on page 3).

Index	Initial Order	After `chunksize == 1`	After `chunksize == 2`	After `chunksize == 4`
0	smee			
1	noodler			
2	turley			
3	cecco			
4	jukes			
5	mullins			
6	whibbles			
7	starkey			

Problem 4: **Trace Functions (20 points)**

Write what will be printed to the screen when each of the following snippets of code is executed in PyCharm or in the Online Python Tutor.

Write your solution in the box provided.
Do not write any scratch-work in the solution box.
In your solution, be very precise with spacing, line breaks, etc.
Treat each sub-problem as an independent question.
All questions in this section are worth 5 points.

Problem 4A

```
def incPrint(a, b):
    a += b
    print(a, b)

a = 5 % 3
b = 10**2
print(a, b)
incPrint(b, a)
incPrint(b, 8)
print(a, b)
```

Problem 4B

```
def appSee(L, a):
    L[a] = max(L)
    print(L)

L = [10, 7, 14, 8]
appSee(L, 1)
appSee(L, -1)
```

Problem 4C

```
def addSum(L):
    L.append(sum(L))
    print(L)

A = [1,2,3]
addSum(A)
addSum(A)
```

Problem 4D

```
def printRowInfo(R):
    print('Items:', len(R))
    for x in R:
       print(len(x),end=' ')
    print()

A = [["134", "44"], ["123"],
     ["4", "35", "1"]]
for i in range(len(A)):
    printRowInfo(A[i])
```

Extra Page . . .

Problem 5: **Defining classes (20 points)**

In this problem, you will define a class to represent an order for a meal. Your class should be named `MealOrder`, and you should define the methods below. Hint: if you are using the `print` or `input` functions to implement these methods, you are doing it wrong. You do *not* need to write any docstrings.

`__init__`: Initializes a `MealOrder` object. Takes one parameter: a customer name. It saves this parameter in an attribute. It also initializes two additional attributes: one is the list of items that are part of the order; the other is the list of items from the order that are complimentary/free. Both lists are initially empty.

`add`: Takes one parameter: a string representing a food. Adds this food to the order.

`contains`: Takes one parameter: a string representing a food. Returns `True` if the food is part of the order, and `False` otherwise.

`exchange`: Takes two parameters: the original item and new item. If the original item is part of the order, it changes it to be the new item (the original item is no longer in the order and the new item is in the order instead) and returns `True`. If the original item is not part of the order, then nothing in the order changes and the method returns `False`.

`to_string`: Returns a string that represents the order. For example, if the order attribute holds the value `['apple', 'pizza', 'milk']` then this method returns the string `'apple, pizza, milk'`. If the order attribute is the empty list `[]`, this method returns the empty string `''`.

`give_free`: Takes one parameter: a string representing a food. If the food is in the order, then it adds the food to the list of complimentary items. If the food is not in the order, then it adds the food to the list of complimentary items and it also adds the food to the order.

Start your solution on the next page...
Toward the end of the exam, there are extra pages if needed.

Problem 5, continued . . .

Problem 6: **Using classes (30 points)**

For this problem, you must write a complete program. This includes logic in `def main()`, a call to `main()`, any necessary library imports, etc. You do *not* need to write any docstrings.

To earn full credit, you must *use the methods* from the `MealOrder` class whenever appropriate. You may assume that the class, as described in Problem 5, has already been correctly implemented for you. Read the instructions carefully before you start coding!

Your program helps run the kitchen on a pirate ship. It should do the following:

1. Define a function called `GetOrder` that does the following:
   - Prompts the user for order information. This information is a string that is the pirate customer's name (always two words), followed by an order (*any number* of one-word food items). For example, DeadEye Pete's order has 3 items:
     `DeadEye Pete  meat rum sea-biscuit`
   - If the user's string is empty, the function returns `None`.
   - If the user's string is too short to be a valid order, the program exits with some error message. The shortest valid order is 3 words long, like: `Redhand Jane rum`.
   - Otherwise, the function creates a `MealOrder` object using the pirate's name and populates the order with the foods that were listed.
   - If the customer has "deadeye" (in any capitalization) as their first or last name, then they get free `'rum'` and free `'turtle-soup'`.
   - Every order on the pirate ship must include either `'rum'` or `'grog'`. If the order lacks these, then add `'grog'` to the order by default.
   - Finally, the function returns the `MealOrder` object.
2. Define a function called `GetAllOrders` that does the following:
   - Repeatedly calls `GetOrder` until it returns `None`; adds each of the returned `MealOrder` objects into a list and then returns that list.
3. Define a function called `main` that does the following:
   - Call `GetAllOrders` to get a list of all the orders.
   - Using the orders, make a list of just the orders that include `'meat'`.
   - There is only enough meat for the first 20 orders! The remaining meat orders need to be modified: for these, exchange `'meat'` for `'slop'`. For each changed order, add complimentary `'rum'` as an apology. In each case, print a summary of these changes, including the old order and new order (using the `to_string` method) and pirate name, matching the below format:

```
Order for Anne Bonny was changed
from: meat, grog, turtle-soup
to: slop, grog, turtle-soup, rum
```

Start your solution on the next page...
Toward the end of the exam, there are extra pages if needed.

Problem 6, continued . . .

Extra Pages . . .

Extra Pages . . .